WIICA

Workload ISA-Independent Characterization for Applications

Yakun Sophia Shao, Emma Wang, Gu-Yeon Wei, David Brooks
Harvard University
Specialized architectures are decoupled from legacy ISAs.

Spectrum of Specialization:
- General-Purpose CPU
- GPU
- Fixed-Function ASIC

Efficiency:
- Low Efficiency
- High Efficiency

Programmability:
- High Programmability
- Low Programmability
Specialized architectures are decoupled from legacy ISAs.

Spectrum of Specialization:

- **General-Purpose CPU**
- **GPU**
- **Fixed-Function ASIC**

- Low Efficiency ↔ High Efficiency
- High Programmability ↔ Low Programmability
- Tied to a Specific ISA ↔ No ISA
Performance-Counter Based Workload Characterization

• Metrics
  – IPC
  – Cache miss rates
  – Branch mis-prediction rates
  – ...

• Microarchitecture-dependent
  – What if there is a bigger cache/a better branch predictor?
  – Not program intrinsic characteristics
ISA impacts program behaviors.

Stack Overhead
- Limited Registers
- Additional Load/Store
ISA impacts program behaviors.

Stack Overhead
• Limited Registers
• Additional Load/Store

Complex Operations
• Memory Operands
• Vector Operations
ISA impacts program behaviors.

Stack Overhead
• Limited Registers
• Additional Load/Store

Complex Operations
• Memory Operands
• Vector Operations

Calling Conventions
Goal:

• An analysis tool to characterize workloads ISA-independent characteristics for specialized architectures
WIICA Summary

Goal:
• An analysis tool to characterize workloads ISA-Independent characteristics for specialized architectures

Methods:
• Leverage compiler’s intermediate representation (IR)
• Categorize characteristics into compute, memory, and control
WIICA Summary

Goal:
• An analysis tool to characterize workloads ISA-Independent characteristics for specialized architectures

Methods:
• Leverage compiler’s intermediate representation (IR)
• Categorize characteristics into compute, memory, and control

Takeaways:
• ISA-dependent characterization is misleading for specialization.
• ISA-independent characterization allows designers to quickly identify opportunities for specialization.
Tool Overview

Program

IR Trace

ISA-Independent

Characterization for Specialized Architecture

Design of Specialized Architecture

Compute  Memory  Control
Tool Overview

Program

IR Trace

x86 Trace

Design of Specialized Architecture

ISA-Independent

Characterization for Specialized Architecture

ISA-Dependent

Compute

Memory

Control

ISA-Dependent
Program Representations

Program

ILDJIT

IR Trace

LLVM

x86 Trace
Program Representations

ILDJIT
- A modular compilation framework
- Performs machine-independent classical optimizations at the IR level
- Uses LLVM’s back end to
  - Do machine-dependent optimizations
  - Generate machine code

Program Representations

ILDJIT IR

- High-level IR
- Machine-, ISA-, and system-library-independent
- Features:
  - 80 instructions
  - Unlimited registers
  - Only loads/stores access memory
  - No vector operations
  - Parameters are passed by variables
Program Representations

x86 Trace
- Used for ISA-dependent analysis
- Semantically equivalent to the IR code
- Collected with Pin instrumentation
Tool Overview

Program

IR Trace

x86 Trace

ISA-Independent

ISA-Dependent

Design of Specialized Architecture

Characterization for Specialized Architecture

Compute

Memory

Control
ISA-Independent Workload Characteristics

- **Compute**
  - Opcode Diversity
  - Static Instructions (I-MEM)

- **Memory**
  - Memory Footprint (D-MEM)
  - Memory Entropy
  - Locality Score

- **Control**
  - Branch Instruction Counts
  - Branch Entropy
ISA-Independent Workload Characteristics

- **Compute**
  - Opcode Diversity
  - **Static Instructions (I-MEM)**

- **Memory**
  - Memory Footprint (D-MEM)
  - Memory Entropy
  - Locality Score

- **Control**
  - Branch Instruction Counts
  - Branch Entropy
Compute::Static Instructions

![Bar chart showing the number of unique static instructions for various benchmarks. The x-axis represents the benchmarks, including 179.art, 183.equake, 188.ammp, 164.gzip, 175.vpr, 175.vpr_2, 181.mcf, 186.crafty, 254.gap, 255.vortex, and 256.bzip2. The y-axis represents the number of unique static instructions, ranging from 0 to 4500. The chart distinguishes between x86 and IR versions of the benchmarks.]
ISA-Independent Workload Characteristics

- **Compute**
  - Opcode Diversity
  - Static Instructions (I-MEM)

- **Memory**
  - Memory Footprint (D-MEM)
  - **Memory Entropy**
  - Locality Score

- **Control**
  - Branch Instruction Counts
  - Branch Entropy
Memory::Entropy

Entropy: a measure of the randomness

\[ Entropy = - \sum_{i=1}^{N} p(x_i) \times \log_2 p(x_i) \]
Memory::Entropy

Entropy: a measure of the randomness

\[ \text{Entropy} = - \sum_{i=1}^{N} p(x_i) \log_2 p(x_i) \]

Case 1:
X is always a constant.

\[ p(X) = 1 \]
\[ \log_2 p(X) = 0 \]
\[ \text{Entropy} = 0 \]

Case 2:
N possible outcomes of X occur equally.

\[ p(X) = \frac{1}{N} \]
\[ \log_2 p(X) = \log_2 N^{-1} \]
\[ \text{Entropy} = -N \cdot \frac{1}{N} \cdot \log_2 N^{-1} \]
\[ \text{Entropy} = \log_2 N \]
## Memory::Global Address Entropy

### Temporal Locality

<table>
<thead>
<tr>
<th>Address Stream A</th>
<th>Address Stream B</th>
</tr>
</thead>
<tbody>
<tr>
<td>(less temporal locality)</td>
<td>(more temporal locality)</td>
</tr>
<tr>
<td>0000</td>
<td>0011</td>
</tr>
<tr>
<td>0001</td>
<td>0011</td>
</tr>
<tr>
<td>0010</td>
<td>0011</td>
</tr>
<tr>
<td>0011</td>
<td>0011</td>
</tr>
</tbody>
</table>

Entropy = 2

Entropy = 0

---

*Yen, Draper, and Hill. Notary: Hardware Techniques to Enhance Signatures. MICRO 08*
Memory::Global Address Entropy

Temporal Locality

Address Stream A (less temporal locality)

Address Stream B (more temporal locality)

Entropy = 2

Entropy = 0

Yen, Draper, and Hill. Notary: Hardware Techniques to Enhance Signatures. MICRO 08
Memory::Global Address Entropy

Temporal Locality

Address Stream A
(less temporal locality)

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1

Entropy = 2

Address Stream B
(more temporal locality)

0 0 1 1
0 0 1 1
0 0 1 1
0 0 1 1

Entropy = 0

Yen, Draper, and Hill. Notary: Hardware Techniques to Enhance Signatures. MICRO 08
## Memory::Local Address Entropy

### Spatial Locality

<table>
<thead>
<tr>
<th>Address Stream A</th>
<th>Address Stream B</th>
</tr>
</thead>
<tbody>
<tr>
<td>(less spatial locality)</td>
<td>(more spatial locality)</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>0 0 1 1</td>
</tr>
</tbody>
</table>

![Graph showing Local Entropy vs # of Bits Skipped]

- **Local Entropy**: The graph illustrates the reduction in local entropy as the number of bits skipped increases.
- **Address Streams**: Address Stream A (less spatial locality) compared to Address Stream B (more spatial locality).
Memory::Local Address Entropy

Spatial Locality

Address Stream A  
(less spatial locality)  
0 0 0 0  
0 1 0 0  
1 0 0 0  
1 1 0 0

Address Stream B  
(more spatial locality)  
0 0 0 0  
0 0 0 1  
0 0 1 0  
0 0 1 1

# of Bits Skipped

Local Entropy

Memory Address Local Entropy

# of Bits Skipped

10

8

6

4

2

0

0 2 4 6 8 10

IR

x86
Memory::Spatial Locality Score

Address Stream A
(less spatial locality)

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>0x0000</td>
</tr>
<tr>
<td>0x0004</td>
<td>0x0001</td>
</tr>
<tr>
<td>0x0008</td>
<td>0x0002</td>
</tr>
<tr>
<td>0x000C</td>
<td>0x0003</td>
</tr>
</tbody>
</table>

Address Stream B
(more spatial locality)

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>0x0000</td>
</tr>
<tr>
<td>0x0004</td>
<td>0x0001</td>
</tr>
<tr>
<td>0x0008</td>
<td>0x0002</td>
</tr>
<tr>
<td>0x000C</td>
<td>0x0003</td>
</tr>
</tbody>
</table>

\[
L_{\text{spatial}} = \sum_{\text{stride}=1}^{\text{stride}<\infty} \frac{P(\text{stride})}{\text{stride}}
\]

\[
L_{\text{spatial}} = \begin{cases} 
P(\text{stride} = 1) & \text{if } \text{stride} = 1 \\
1 & \text{if } \text{stride} < \infty
\end{cases}
\]

\[
L = \frac{P(\text{stride} = 4)}{\text{stride} = 4} = \frac{1}{4}
\]

\[
L = \frac{P(\text{stride} = 1)}{\text{stride} = 1} = 1
\]

Weinberg, et all, Quantifying Locality in the memory access patterns of HPC applications, In SC, 2005
ISA-Independent Workload Characteristics

- **Compute**
  - Opcode Diversity
  - Static Instructions (I-MEM)

- **Memory**
  - Memory Footprint (D-MEM)
  - Memory Entropy
  - Locality Score

- **Control**
  - Branch Instruction Counts
  - **Branch Entropy**

*Yokota, et al, Introducing Entropies for Representing Program Behavior and Branch Predictor Performance, 07*
Control::Branch Entropy
Tool Overview

Program → Design of Specialized Architecture

IR Trace → ISA-Independent → Characterization for Specialized Architecture

x86 Trace → ISA-Dependent → Compute, Memory, Control
Publications

• Implications of the Power Wall: Dim Cores and Reconfigurable Logic

• ISA-Independent Workload Characterization and its Implications for Specialized Architectures
  – Shao and Brooks, ISPASS, 2013